RECEIVED
CENTRAL FAX CENTER
JAN 2 6 2006

Amendments to the Claims:

1. (Currently Amended) The use of multiple threads in association with a network processor and accessible data available in a tree search structure, including the steps of:

a) providing multiple instruction execution threads as independent processes in a sequential time frame;

b) queuing the multiple execution threads to have overlapping access to the accessible data available in said tree search structure;

c) executing a first thread in a queue;

d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall; and

e) when the stall is due to a short latency event, returning control to the first thread when the event is completed,

and

~~if~~ when the stall is due to a ~~short~~ long latency event, ~~and~~ retaining full control by the next thread until the next thread becomes blocked ~~if the stall is due to a long latency event.~~

2. (Canceled)

3. (Previously presented) The use of multiple threads according to claim 1 wherein a processor instruction is encoded to select a short latency event.

4. (Canceled)

5. (Previously Presented) The use of multiple threads according to claim 1 wherein a processor instruction is encoded to select a long latency event.

6. (Previously Presented) The use of multiple threads according to claim 1 including queuing the threads to provide rapid distribution of access to shared memory.

7. (Original) The use of the multiple threads according to claim 1 wherein the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor.

8. (Original) The use of the multiple threads according to claim 1 further including the step of providing a separate instruction pre-fetch buffer for each execution thread, and collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized.

9. (Original) The use of the multiple threads according to claim 1 wherein the threads are used with zero overhead to switch execution from one thread to the next.

10. (Original) The use of the multiple threads according to claim 9 wherein each thread is given access to general purpose registers and local data storage to enable switching with zero overhead.

11. (Currently amended) A network processor that uses multiple threads to access data available in a tree search structure, including:

a) a CPU configured with multiple instruction execution threads as independent processes in a sequential time frame;

b) a thread execution control for

1) queuing the multiple execution threads to have overlapping access to the data to be accessed;

2) executing a first thread in the queue; and

3) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall, said thread execution control including control logic responsive to the type of event causing the execution to stall that a) temporarily transfers control to the next thread when execution stalls due to a short latency event and returns control to the first thread when the short latency event is completed, ~~or~~ and b) transfers full control of the execution to the next thread when execution of the first thread stalls due to a long latency event with execution continuing on the next thread even after the long latency event is completed, until the next thread becomes blocked.

12. (Canceled)

13. (Currently amended) The processor ~~processing system~~ according to claim ~~12~~ 11 wherein a processor instruction is encoded to select a short latency event.

14. (Canceled)

15. (Currently amended) The processor ~~processing system~~ according to claim ~~14~~ 11 wherein a processor instruction is encoded to select a long latency event.

16. (Currently amended) The processor ~~processing system~~ according to claim 11 including means to queue the threads to provide rapid distribution of access to shared memory.

17. (Currently amended) The processor ~~processing system~~ according to claim 16 wherein the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor.

18. (Currently amended) The processor ~~processing system~~ according to claim 11 further including a separate instruction pre-fetch buffer for each execution thread, and means for collecting instructions in a prefetch buffer for an idle execution thread when the instruction bandwidth is not being fully utilized.

19. (Currently amended) The processor ~~system~~ according to claim 11 wherein the processor uses zero overhead to switch execution from one thread to the next.

20. (Currently amended) The processor ~~system~~ according to claim 19 wherein each thread is given access to an array of general purpose registers and local data storage to enable switching with zero overhead

21. (Currently amended) The processor ~~system~~ according to claim 20 wherein the general purpose registers and the local data storage are made available to the processor by providing one address bit under the control of the thread execution control logic and by providing the remaining address bits under the control of the processor.

22. (Currently amended) The processor ~~system~~ according to claim 20 wherein the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which array will be delivered to the processor for a given thread.

23. (Currently amended) The processor ~~system~~ according to claim 20 wherein the local data storage is fully addressable by the processor, an index register is

contained within the register array, and the thread execution control has no address control over the local data storage or the register arrays.

24 - 26 (Canceled)

27. (Currently amended) A thread execution control useful for the efficient execution of independent threads comprising:

a) a priority FIFO buffer for storing thread numbers; said buffer including

1) means for loading a thread number into the buffer when a packet is dispatched to the processor;

2) means for unloading a thread number from the buffer when a packet has been enqueued for transmission;

3) thread number transfer from highest priority to lowest priority in the buffer when the thread with the highest priority encounters a long latency event ~~occurs~~, and

4) thread outlets of the buffer used to determine priority depending on the length of time a thread has been in the buffer

b) a plurality of thread control state machines, ~~one for each thread~~ for use in shifting execution control between threads upon occurrence of latency events; and

c) an arbiter for determining the thread execution priority among multiple threads based upon signals outputted from the FIFO buffer and the state machines.

28. (Canceled)

29. (Original) The thread execution control according to claim 27 wherein the arbiter controls the priority of execution of multiple independent threads based on the Boolean expression:

$$G_n = R_n \cdot \{(P_A = n) + R_{PA} \cdot (P_B = n) + R_{PA} \cdot R_{PB} \cdot (P_C = n) \cdots\}$$

where: G is a grant

R_n is a request from a given thread;

P_A, P_B and P_C represent threads ranked by alphabetical subscript according to priority;

$_n$ is a subscript identifying a thread by the bit or binary number comprising

a) determining whether a request R is active or inactive;

b) determining the priority of the threads;

c) matching the request R with the corresponding thread P; and

d) granting a request for execution if the request is active and if the corresponding thread P has the highest priority.

30. (Previously Presented) The thread execution control according to claim 27 wherein the thread control state machine comprises control logic to:

(a) dispatch a packet to a thread;

(b) move the thread from an initialize state to a ready state;

(c) request execution cycles for the thread;

(d) move the thread to the execute state upon grant by the arbiter of an execution cycle;

(e) continue to request execution cycles while the thread is queued in the execute state; and

(f) return the thread to the initialize state if there is no latency event, or send the thread to the wait state upon occurrence of a latency event.

31. (Currently amended) The thread ~~executing~~ execution control according to claim 27 wherein the FIFO buffer further includes ~~means~~ control logic to detect occurrence of latency events, and to differentiate between a short latency event and a long latency event.

32. (Previously Presented) The method of Claim 1 wherein the short latency event includes a time interval of twenty-five or less machine cycles wherein a machine cycle is approximately between 5 and 7.5 nanoseconds.

33. (Previously Presented) The method of Claim 1 wherein the long latency event includes a time interval greater than twenty-five machine cycles wherein a machine cycle is approximately between 5 and 7 nanoseconds.

34. (Canceled)

35. (New) The thread execution control according to claim 27 further including a prefetch buffer for each independent execution thread.